AMENDMENTS TO THE SPECIFICATION

In the Specification:

Please replace the title with the following:

Title: ARCHITECTURE AND METHOD FOR SERIALIZATION AND DESERIALIZATION OF OBJECTS

Please replace the paragraph on page 1 beginning at line 11 with the following amended paragraph:

Serialization is a mechanism for taking a data structure (*e.g.*, a graph of objects, a call stack) and converting it to a stream of data in a particular external format. At deserialization, the stream of data is converted back into the data structure with the same topology of the original data structure. Serialization facilitates inter-process communication such as transmissions between address spaces, or persistence such as storage on a non-volatile media. For example, an executable program can be serialized at a server end and transferred over to an application running on a client end for executing the executable program at the client. Additionally, an object can be serialized and stored in contiguous memory to save space. Inter-process communications and object persistence are fundamental techniques used in many software applications. The main use of a serialization stream is for saving a graph of objects externally in a file or transferring the serialization data by remoting between computer processes or processors.

Please replace the paragraph on page 2 beginning at line 1 with the following amended paragraph:

Serialization is typically performed by a formatter. Serialization involves writing the state information of parameters and/or objects to a stream that may be read by a formatter on a server, so that the parameters and/or objects can be recreated on the server side. Similarly,

serialization involves writing the state information of return parameters and/or objects to a stream that may be read by a formatter on the client side, so that the return parameters and/or objects can be recreated back on the client side. Formatters also typically perform the inverse operation of deserialization. The formatters dictate the format in which the parameters and/or objects and their associated state are written to the stream. For example, a first formatter may employ a binary format while a second formatter may employ an XML format to represent the types of ~~the~~ parameters and/or objects and their associated state. Conventionally, formatters are not selectable, pluggable or customizable and thus conventional systems suffer from problems associated with inflexibility (*e.g.*, inability to interact with new externalized formats).

Please replace the paragraph on page 7 beginning at line 21 with the following amended paragraph:

Alternatively, a third party object or file (*e.g.*, a surrogate) will determine what serialization information ~~that~~ will be provided for a given object type to the pluggable formatter 20. The third party object or file can provide the pluggable formatter 20 with serialization information about an object type. A surrogate is an object that specifies what information is serialized for an object of a particular type. An object or third party object can also provide serialization information about an object reference to be serialized containing information about the object type, as opposed to serializing an instance of the object itself. The object reference can be employed in remoting situations when a proxy is to be provided to a client or when an object type is restricted to one instance per process, such that only one copy of the object type can be at any one location at a time.

Please replace the paragraph on page 8 beginning at line 11 with the following amended paragraph:

The serialization and deserialization portion 14 then provides the pluggable formatter 20 with the serialization information. The pluggable formatter 20 converts the serialization information into a serial stream 22 in a particular externalized format defined by the pluggable formatter 20. For example, the pluggable formatter can write the serial stream 22 in XML,

HTML, binary or a variety of other externalized formats. A variety of different formatters can be employed on the system 10[[,]] so that a developer can select between different externalized formats in which to serialize the object graph 18 into the ~~serialized~~serial stream 22.

Please replace the paragraph on page 8 beginning at line 19 with the following amended paragraph:

During deserialization, the pluggable formatter 20 decodes the serial stream 22 and the serialization and deserialization portion 14 works in conjunction with the utility and contract portion 16 to deserialize the serialized stream into a graph of objects. The main function of the utility and contract portion 16 is to maintain ~~the~~ invariants during the deserialization process. The pluggable formatter 20 receives the serial stream 22 in an encoded/serialized format in which the pluggable formatter 20 is designed to convert into object form (*e.g.*, decode). The utility and contract portion 16 works in conjunction with the pluggable formatter 20 to repopulate the object graph 18. The pluggable formatter 20 is responsible for enforcing some of the type safety provisions and requesting an object of a particular type from the utility and contract portion 16. The utility and contract portion 16 instantiates an unitialized instance of a given object type for populating with object data for each object being deserialized. The utility and contract portion 16 determines the type to be instantiated. The utility and contract portion 16 then tracks objects and ~~provided~~provides fixups due to forward object references. In the case of remoting, the utility and contract portion is responsible for swapping objects (*e.g.*, marshal/proxy) and getting the real object (*e.g.*, objectreference/realobject).

Please replace the paragraph on page 9 beginning at line 25 with the following amended paragraph:

The serialization selector 36 can be part of the formatter 44 or part of the serialization architecture. The serialization selector 36 matches a rule set 32 out of a plurality of rule sets 34 to a particular object type or informs the formatter 44 where to find a rule set for the particular object type. The object can contain information that associates that object with a rule set and invokes methods of the rule set, for example, by implementing an interface or through

inheritance using a base class. In this way, the object can employ methods associated with a rule set and define a customized rule set within the object for determining the serialization information that will be provided to the formatter 44 for a given object type. Alternatively, the serialization selector 36 can identify customized rule sets in other objects for a given object type, so that a user can define serialization information for a given type within a third party object. For example, the customized rule set can reside in a third party object defining the serialization information that will be provided for a given object type and this rule set is used to determine ~~that~~the serialization information that will be provided from an object of that type. Alternatively, a customized rule set can be provided to define serialization information about a reference object type for a given object type which ~~contain~~contains information about the serialized object but not the object data itself. This can apply to value types that are stack allocated as well. The customized rule set for implementing an object type reference can be defined in the object or in a third party object.

Please replace the paragraph on page 10 beginning at line 13 with the following amended paragraph:

Some objects have information that needs to be secured or cannot be serialized in a standard fashion. Therefore, one of the rule sets 32, 34 can be provided to define how the object would be serialized or to define how the object serialization information within the portion of the serialized stream containing the object information would get populated with object data and what object data ~~would~~ the serialization stream would receive. The rule set can reside within the object or within another object based on a given object type. Serialization information is essentially a series of name value pairs representing fields of the objects with additional information contained in metadata such as object type and data type. One of the rule sets 32,34 can dictate if serialization information is to be provided about another object type in place of that object for a given object type or restrict the information that will be provided for that object for a given object type.

Please replace the paragraph on page 11 beginning at line 16 with the following amended paragraph:

Object C and object D are special cases where the rule set determines that serialization information from another object will be serialized in place of object C and object D. Object C is an object that is restricted to one instance per process. In this circumstance, a user can define a rule set that provides an object or object reference type F that contains information about object C, that is needed for deserialization of F ~~an~~and ultimately retrieval of object C. Object D is a marshall object for providing a remoting interface to a client. In this circumstance, a rule set can be defined that provides a marshal reference G that contains information about object D and E, that is needed for creating a proxy reference at a client end.

Please replace the paragraph on page 11 beginning at line 25 with the following amended paragraph:

After it is determined how the serialization information of the modified graph 42 is going to be populated with object information, the object information is then pushed to the data structure 50. This is repeated for each object in the modified graph 42, until object information for all of the objects have been written to the data structure 50. The data structure 50 is then transmitted to the pluggable formatter portion 48 of the formatter 44. The pluggable formatter portion 48 serializes the graph of objects in a preselected externalized format. Alternatively, the objects can be serialized as they are retrieved (*e.g.*, on the fly), such that serialization information for each object is pushed to the data structure 50 one at a time and then serialized by the pluggable formatter [[44]]portion 48 in a preselected externalized format. This is then repeated for each object until the entire graph 42 has been serialized.

Please replace the paragraph on page 12 beginning at line 15 with the following amended paragraph:

The object can contain information that associates that object with a rule set and invokes methods of the rule set, for example, by implementation of an interface. In this way, the object

can employ methods associated with a rule set and define a customized rule set within the object for determining the deserialization information that will be provided to the formatter 80. Alternatively, the object can identify a customized rule set for obtaining serialization information from another object, so that the user can define deserialization information that informs a set of deserialization services 70 that the object being received is a reference object which ~~contain~~contains information about the object but not the object data itself. The serialization selector can also determine that a third party object defines the serialization information that is provided for an object of a given type.

Please replace the paragraph on page 12 beginning at line 25 with the following amended paragraph:

The set of deserialization services 70 contain an object type service 72 which determines what object type is to be provided for deserialization of the object read off the serial stream. The object type service 72 then creates an uninitialized instance or object shell of that type, a similar type or another type of which no constructor has been called for deserialization of the object. The constructor itself can reside in the object or serialized data and can be invoked once the object is deserialized in certain circumstances. An object tracking service 76 is provided for tracking objects during the deserialization process. Each object registers with the object tracking service 76 as it is deserialized. The object tracking service 76 looks at the object and determines references to other objects within the object (*e.g.*, forward references, backward references). If the referenced object is not available (*e.g.*, down the serialization stream), the formatter 80 commands the object tracking service 76 to record a fixup for that object. The object tracking service 76 then records a fixup for that object. The object tracking service 76 also guarantees that the deserialization information is complete before an object can be called. Once the deserialization of one or all objects is complete, a fixup service 74 is performed to fill in objects with data associated with the forward references to complete reinstantiation of the object or the graph 86.

Please replace the paragraph on page 15 beginning at line 14 with the following amended paragraph:

Objects that wish to control their own serialization can do so by implementing a serializable interface 116 in the class. The serialization interface 116 provides methods for allowing a class author to specify how the object is to be serialized within the object. If a class author wants the object to be serializable, the class author can mark that object as being serializable or provide serializable attributes in the object. The class author can also ~~marks~~mark portions of the objects with non-serializable attributes so that these portions do not get serialized. The formatter 132 will then call a formatter services component 124, which will employ a default serialization for the object. The serializable interface 116 allows the class author to specify the data transmitted on the stream and control how the object is reinstantiated based on that data. For example, the serializable interface 116 can be an interface that specifies only one method (*e.g.*, GetObjectData) but also implies the existence of a constructor with a certain signature. The method is called by the formatter 132, the formatter services 124 or the formatter interface 115 during serialization.

Please replace the paragraph on page 15 beginning at line 28 with the following amended paragraph:

At that point, the object is responsible for adding the set of information required to specify its state to the serialization information as a set of name-value pairs. The user is also responsible for providing the object type as an additional form of metadata. Serialization information is a set of name-value pairs (*e.g.*, a set of tuples of name, type, value) that contains the information necessary to serialize or deserialize an object. At serialization time, the object is responsible for inserting its values into the information. At deserialization time, it can read out the same name-value pairs. Any objects added to the serialization information are automatically tracked by the formatter 132 for later deserialization. If any object in the graph 130 is not serializable, the formatter interface ~~124~~115 will fail serialization.

Please replace the paragraph on page 17 beginning at line 4 with the following amended paragraph:

An object can be changed into a remote object by implementing a marshal by reference (MarshalByRef) interface 114 into the class. In remoting scenarios, a single surrogate dictates the data that is transmitted for every object which extends the MarshalByRef interface 114. An object can also include an object that implements an object reference (ObjectRef) interface 112, so that serialization information can be provided on the object reference type instead of the object type itself. For example, an object employing the serializable interface 116 can change its type to an object reference type. The object reference type can then reside in another class that provides serialization information for the object reference type. The object reference type then contains information on the object and how to retrieve the actual object when it is safe to do so.

Please replace the paragraph on page 19 beginning at line 6 with the following amended paragraph:

Once the type is determined, the formatter services component 176 creates an uninitialized instance of that type. No constructor is ever called on this object, except for objects implementing a serializable interface ~~162~~163. The object is registered with an object manager component 174 along with its object ID. The deserialization decision loop 186 then checks if this type has a surrogate selector 168 to determine if the surrogate selector 168 wishes to control the deserialization of objects of that type. If the surrogate selector 168 does exist and does wish to control deserialization, serialization information of the object is read from the ~~serialization~~serial stream 182 and the surrogate 156 is called. The surrogate implements a serialization surrogate interface 164. If the type implements a serializable interface ~~164~~163, the serialization information of the object is read from the ~~serialization~~serial stream 182 and the methods of the serializable interface ~~162~~163 is called within the object. Otherwise, the serialization information of the object is read from the ~~serialization~~serial stream 182 and the default serialization is employed.

Please replace the paragraph on page 21 beginning at line 12 with the following amended paragraph:

Fig. 7 illustrates another particular methodology for serialization of a graph of objects into a serialization stream in accordance with one aspect of the present invention. The methodology begins at 300 where serialization is invoked to serialize a graph of objects. At 310, an object is retrieved from an object graph. At 320, the method determines if the object has been seen before. If the object has been seen before (YES), the method proceeds to 330. If the object has not been seen before (NO), the method advances to 325 where the object is assigned an object ID. At 330, the methodology determines the serialization rules for that object from a plurality of serialization rules. For example, the method can determine if a surrogate selector is available for objects ~~if~~of this type and a surrogate is provided that handles the serialization information. Alternatively, the object may implement the serializable interface and define its own serialization rules. Finally, a default serialization may be employed if the object is marked with serializable attributes but does not have a surrogate or implement a serializable interface. Furthermore, the serialization rule may restrict object information from being serialized, specify a different object type to be serialized or specify that the object is a remote object type.

Please replace the paragraph on page 22 beginning at line 3 with the following amended paragraph:

Fig. 8 illustrates one particular methodology for determining serialization rules for an object in accordance with one aspect of the present invention. The methodology begins at 400 where rule determination begins. At 410, the type of object is determined and gotten. Based on the object type, a determination is made ~~on~~ whether or not the object has a surrogate at 420. If the object has a surrogate (YES), the methodology proceeds to 425. At 425, the surrogate and the serialization rules from the surrogate are retrieved. The rule determination then ends at 460. If an object has a surrogate it does not need to be marked as serializable. If the object does not have a surrogate (NO), the methodology proceeds to 430. At 430, the methodology determines if the object is serializable. If the object is not serializable (NO), the method proceeds to 435 to return an error. The rule determination then ends at 460. If the object is serializable (YES), the

methodology advances to 440. At 440, a determination is made of whether serialization rules are defined in the object. If the serialization rules are defined in the object (YES), the method proceeds to 445 to retrieve the serialization rules from the object. If the serialization rules are not defined in the object (NO), the method proceeds to 450 to retrieve the default serialization rules. The rule determination then ends at 460.

Please replace the paragraph on page 22 beginning at line 19 with the following amended paragraph:

Fig. 9 illustrates one particular methodology for deserialization of a serialization stream into a graph of objects in accordance with one aspect of the present invention. The methodology begins at 500 where deserialization is invoked to deserialize a serialization stream into a graph of objects. The serialization stream is a linear collection of objects. At 510, an object is retrieved from a serialization stream and registered with an object manager or the like. At 520, the methodology determines what object type is to be populated with object information. For example, the object type can be the same type as the type ~~off~~of the serialization stream or another type defined by the user, for example, by employing a serialization binder service. At 530, an uninitialized instance of the selected object type is provided for deserialization.

Please replace the paragraph on page 22 beginning at line 29 with the following amended paragraph:

At 540, the methodology determines if the object has any objects it references that have not been seen previously. If the object does not have any objects that it references that have not been seen previously (YES), the method proceeds to 550. If the object does have objects that it references that have not been seen previously (NO), the method advances to 545 where the object registers fixups with the object manager corresponding to any objects it references that are not currently available. The object manager can perform fixups to the object graph in parallel with the object being registered once the objects that are referenced become available. However, fixups on object reference types (*e.g.*, object implementing the object reference interface) cannot be performed until the real object is retrieved. The methodology then proceeds to 550. At 550,

the methodology determines the serialization rules for that object from a plurality of serialization rules.

Please replace the paragraph on page 27 beginning at line 8 with the following amended paragraph:

The system 700 includes a communication framework 750 that can be employed to facilitate communications between the clients 710 and the servers 730. The clients 710 are operably connected to one or more client data stores ~~715~~720 that can be employed to store information local to the clients 710 (*e.g.*, pluggable formatters, serialization architecture). Similarly, the servers 730 are operably connected to one or more server data stores 740 that can be employed to store information local to the servers 730 (*e.g.*, pluggable formatters, serialization architecture).